Model Report

Team: Tassay aka.MARS
Team members: Makhmud, Aruay, Ruana, Sanzhar.

I. Project Introduction and Motivation

The main task of our project is multi-class classification. We were originally looking for datasets under torchvision that had additional features (such as bounding boxes and landmarks) we could use to potentially increase the complexity of our project, apart from raw images and labels. During this process, we encountered several problems:

a. The first dataset we selected and started working on turned out to be separate from the torchvision dataset list. This led to a waste of a few hours of EDA and data preprocessing.
b. After a second round of dataset search, we stopped at the INaturalist dataset of 11 super-classes and several hundreds of subclasses. However, even the mini version of its train set was 44.6 GB, which took up all of our disk space and was not feasible for extensive training and testing.
c. Finally, we settled on the Pascal VOC 2012 dataset with 20 classes.

II. Dataset

Name: PASCAL VOC 2012
Link:
https://docs.pytorch.org/vision/stable/generated/torchvision.datasets.VOCDetection.html#torchvision.datasets.VOCDetection

We used the PASCAL VOC 2012 Detection dataset, provided via the torchvision.datasets.VOCDetection interface. This dataset is a standard benchmark for object detection tasks and includes:

- 20 object categories, such as person, car, dog, cat, aeroplane, etc.
- Around 4200 annotated images (for both train and validation)
- Each image contains one or more objects, annotated with:
  Bounding boxes (xmin, ymin, xmax, ymax)
  Class labels per object
  Additional metadata such as image size and segmentation info

- The dataset supports two official splits:
  train (training set)
  val (validation set)

We accessed this dataset using the PyTorch torchvision.datasets.VOCDetection class, which automatically downloads and parses the dataset and corresponding XML annotation files.

The dataset includes the following 20 classes: *[aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike, person, pottedplant, sheep, sofa, train, tvmonitor]*

III. Project Overview

The dataset is inherently multi-label: each image can contain multiple objects from different classes, meaning that a single image may have multiple valid labels.

We initially attempted to tackle this using multi-label binary classification, where the model predicts a binary vector indicating the presence or absence of each class. However, this approach proved to be complex and unstable due to class imbalance, leading to a difficulty in achieving consistent performance across all classes.

To simplify the problem and improve training effectiveness, we decided to leverage the bounding box annotations provided in the dataset. Using these, we cropped out individual objects from each image and assigned a single class label to each cropped image, essentially creating our own version of the dataset with single-label classification.

This modification increased the dataset size and allowed us to train and evaluate standard image classification models more effectively. All models in the project were trained and tested using this cropped, single-label dataset.

We used the following models to train and test on our dataset:

a. Custom CNN
   We trained a simple 2D CNN with 3 convolution layers and ReLU() activations and MaxPooling after each.
   CrossEntropyLoss() was used as the loss function for multi-class classification alongside Adam optimizer with learning rate 0.001 and batch size 32.

   The same model architecture was run for different dataset configurations:
   - Initially, we ran it over the original data with no image modifications other than normalization and resizing.
   - Then, we cropped the dataset using the provided bounding boxes and ran the CNN on it.
   - When we noticed that one class label ('person') had much higher frequency than other labels, we removed some of the 'person' images from the dataset to make its frequency equal to the average over the classes. This made the data more balanced and we ran CNN over this version as well.
   - Afterwards, we incorporated image segmentation (background and foreground) for the same CNN configuration.
   - Lastly, we did a hyperparameter search for the learning rates, optimizer (Adam and SGD), and batch size.

b. ResNet50
c. MobileNet_v3_large
d. ShuffleNet_v2_x1_0
e. EfficientNet_b0
f. SqueezeNet1_1

g. Wide_ResNet50_2
h. ConvNext_tiny

IV. Team Roles and Responsibilities

Makhmud: Streamlit and presentation, team captain
Aruay: EDA, image segmentation, training the CNN on segmented images, hyperparameters search, report
Ruana: EDA, preprocessing, training models, fine-tuning(2 ways), photograph
Sanzhar: Custom CNN modeling, hyperparameters search, domain knowledge, emotional support of the team

V. Problems Encountered

During the course of this project, we encountered several practical and technical challenges:

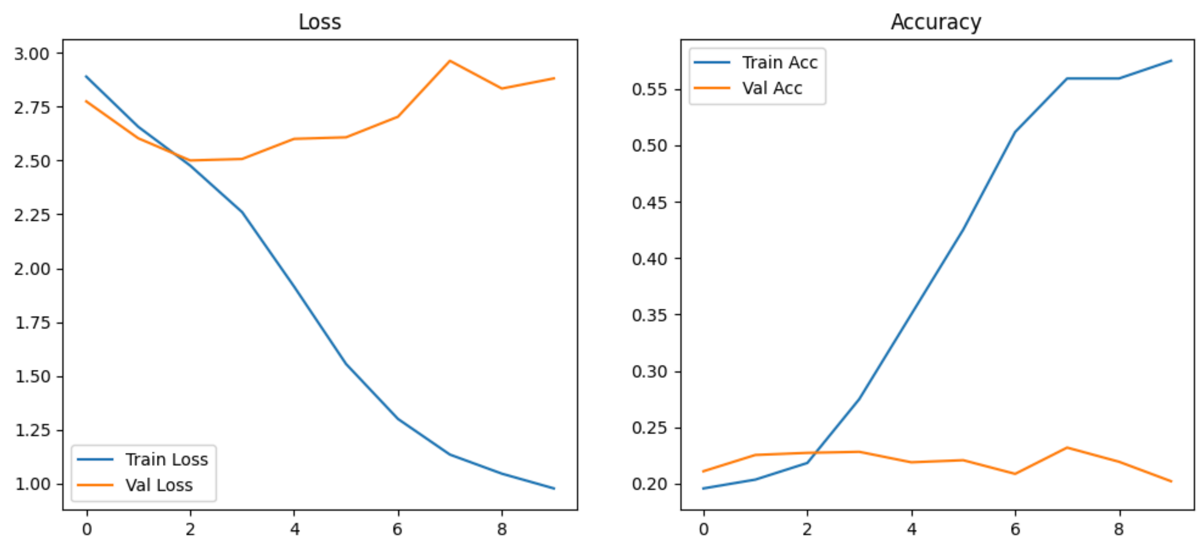
1. Underestimated time constraints
   We initially underestimated the time required for training deep learning models and resolving unexpected issues. Limited access to GPUs on both Kaggle and Google Colab, including timeouts, disconnections, and queue delays, significantly slowed down experimentation and hyperparameter tuning.

2. Dataset size and access
   The datasets we tried first were too large. Even when we only needed a portion of it for our classification tasks, we were still required to download the entire dataset first. This was time-consuming and storage-intensive, particularly on platforms with limited disk space.

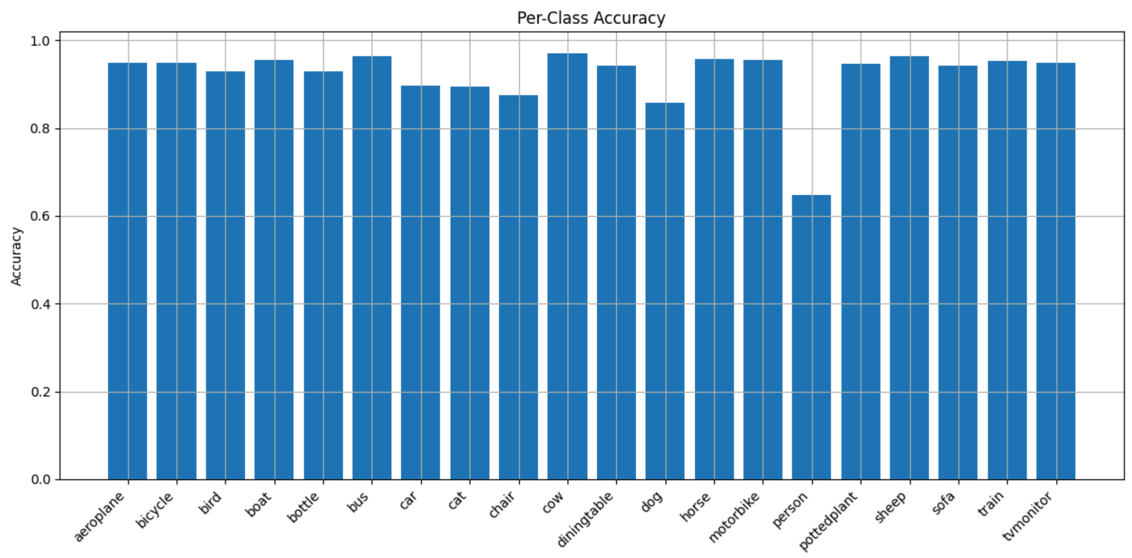
3. Kaggle restrictions on torchvision downloads
   Kaggle environments did not support direct dataset downloads through the `torchvision.datasets` API (e.g., `VOCDetection`), which made it difficult to automate dataset setup. As a result, we had to manually upload pre-downloaded versions to our Kaggle workspace or switch to Google Colab for certain steps.

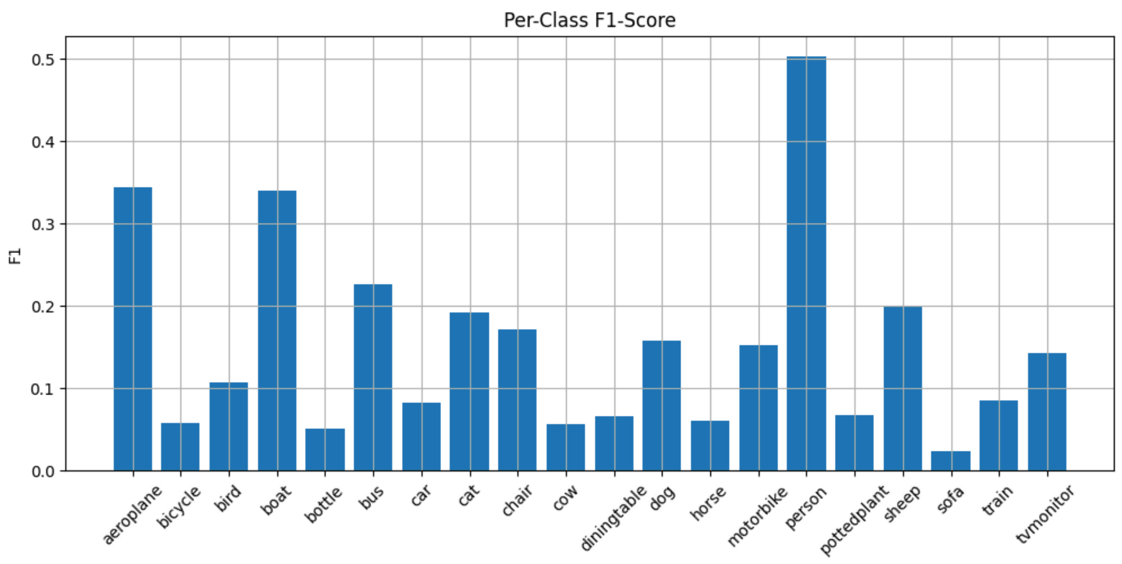
VI. Results and Discussion

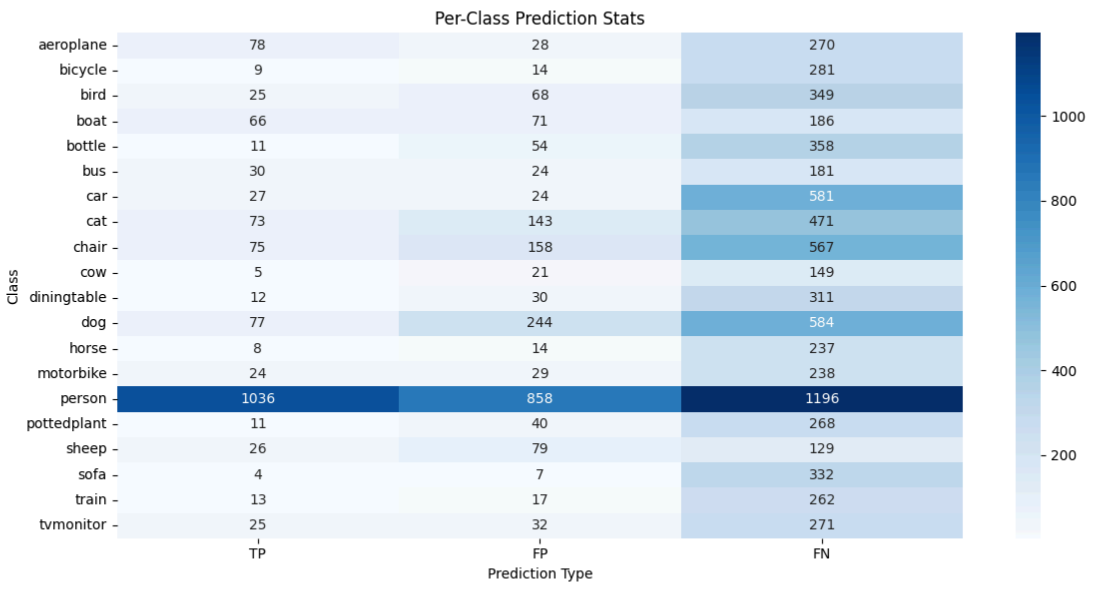
For the custom CNN model, passing the original dataset through it had the lowest validation accuracy at around 22% (random guess would yield 5%).



Multi-label binary classification generated high accuracy but low F1 scores. This was due to high label imbalance: there was a significantly higher number of “negative” labels in comparison to “positive”.

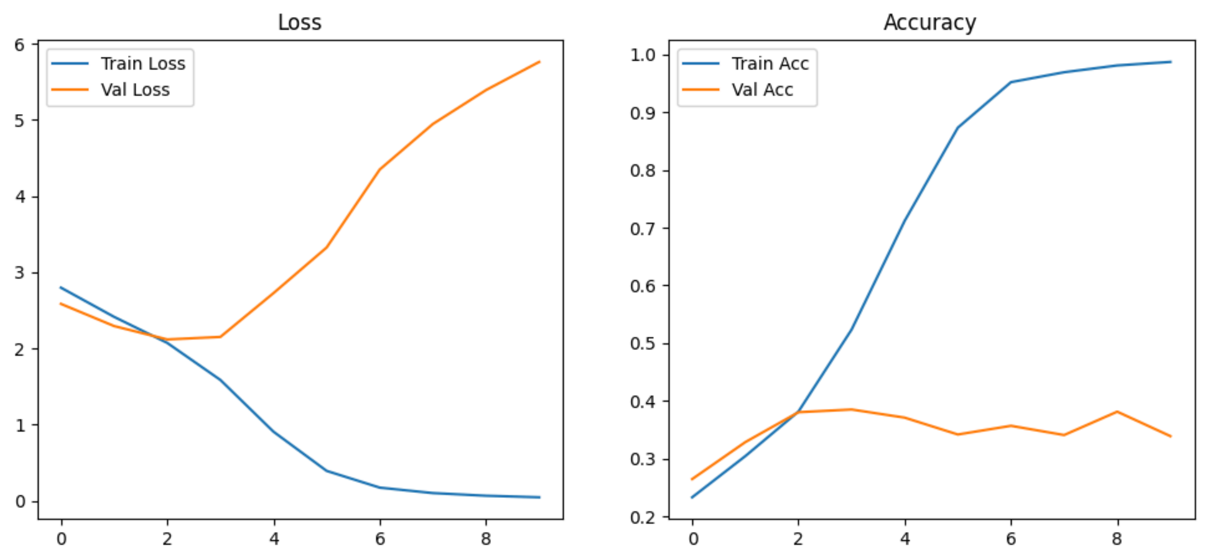


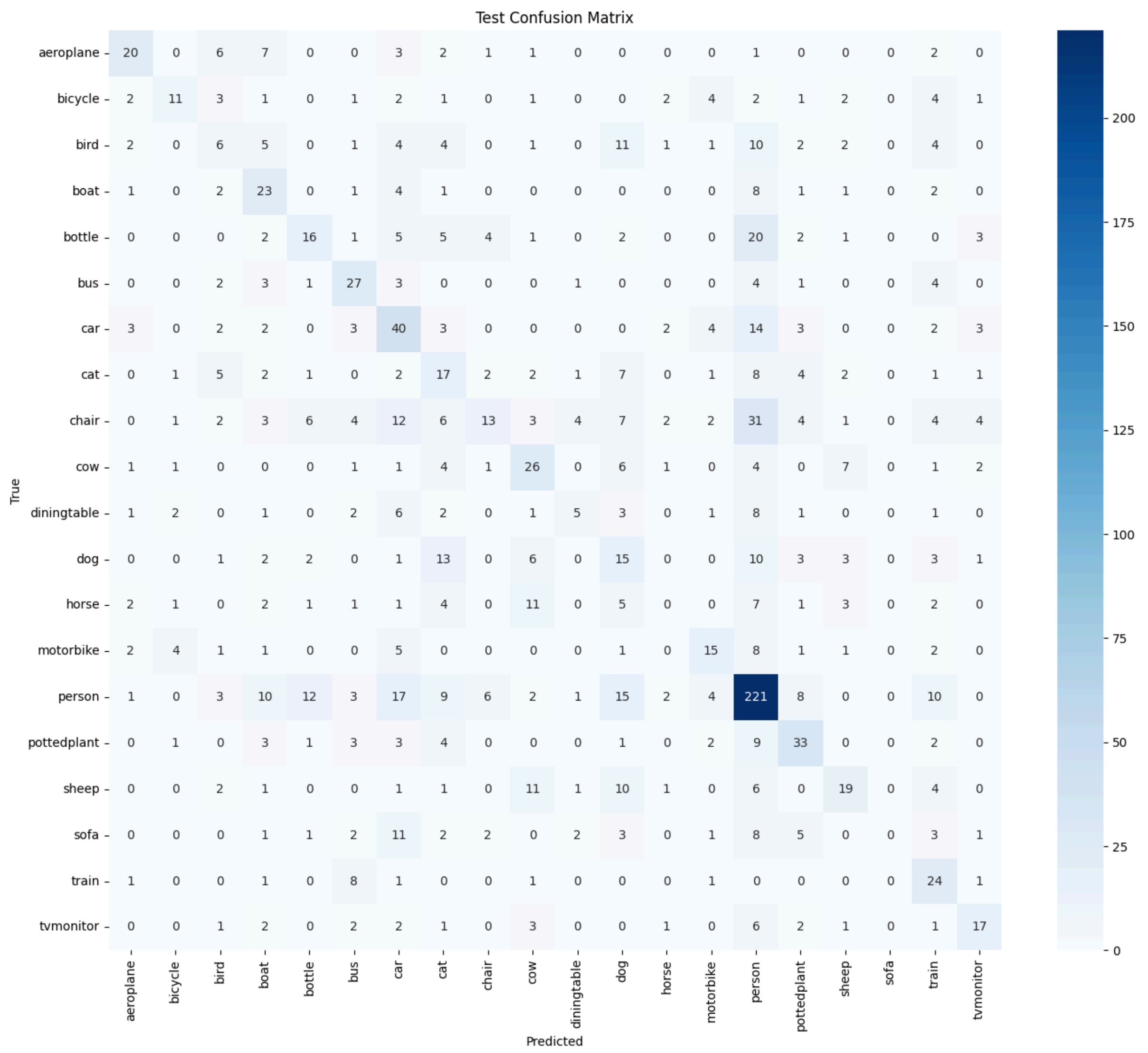




Class	TP	FP	FN
aeroplane	78	28	270
bicycle	9	14	281
bird	25	68	349
boat	66	71	186
bottle	11	54	358
bus	30	24	181
car	27	24	581
cat	73	143	471
chair	75	158	567
cow	5	21	149
diningtable	12	30	311
dog	77	244	584
horse	8	14	237
motorbike	24	29	238
person	1036	858	1196
pottedplant	11	40	268
sheep	26	79	129
sofa	4	7	332
train	13	17	262
tvmonitor	25	32	271

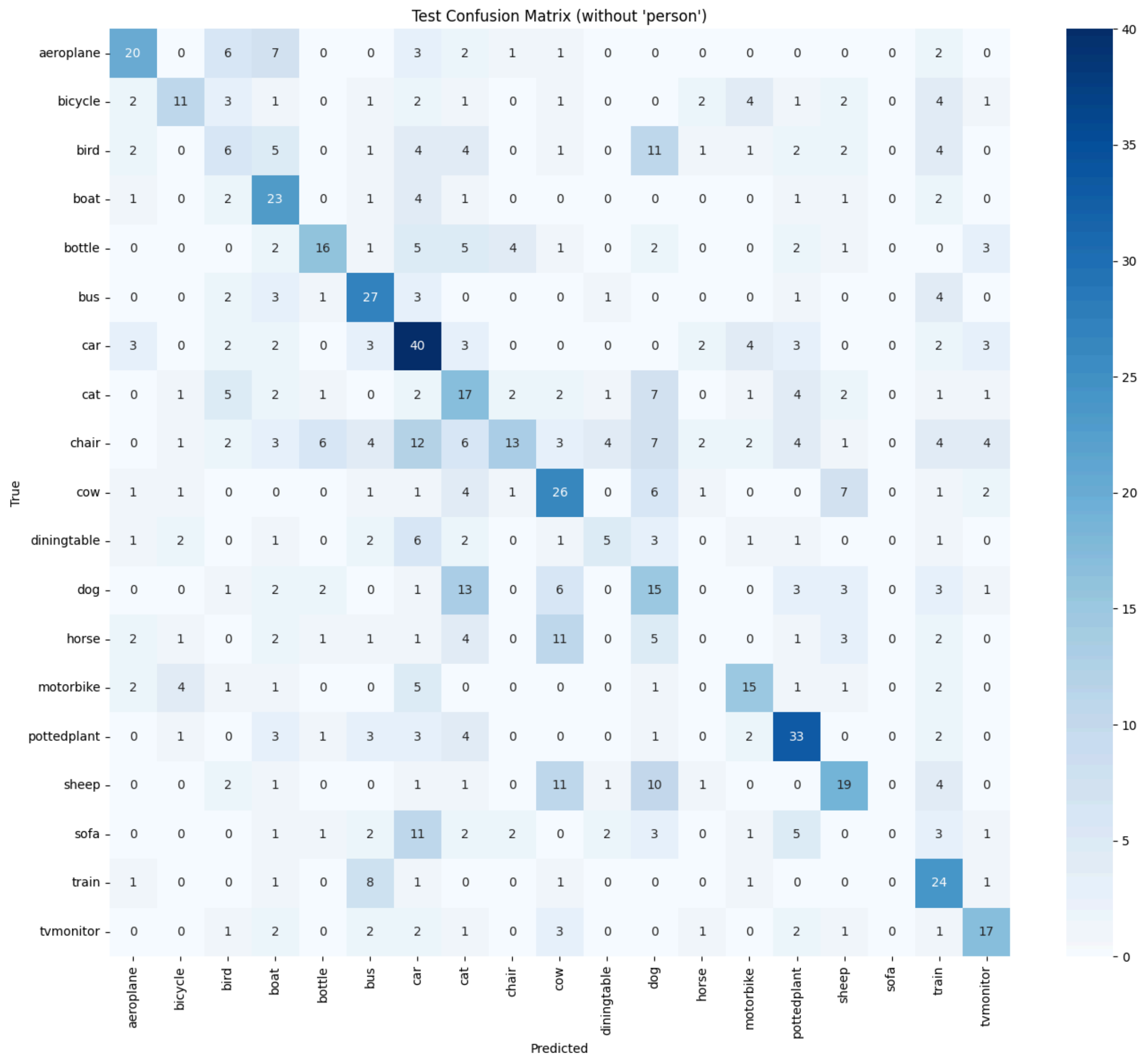
Cropping the dataset using the bounding boxes and making it a simple multi-class classification problem increased both validation accuracy and F1 score.





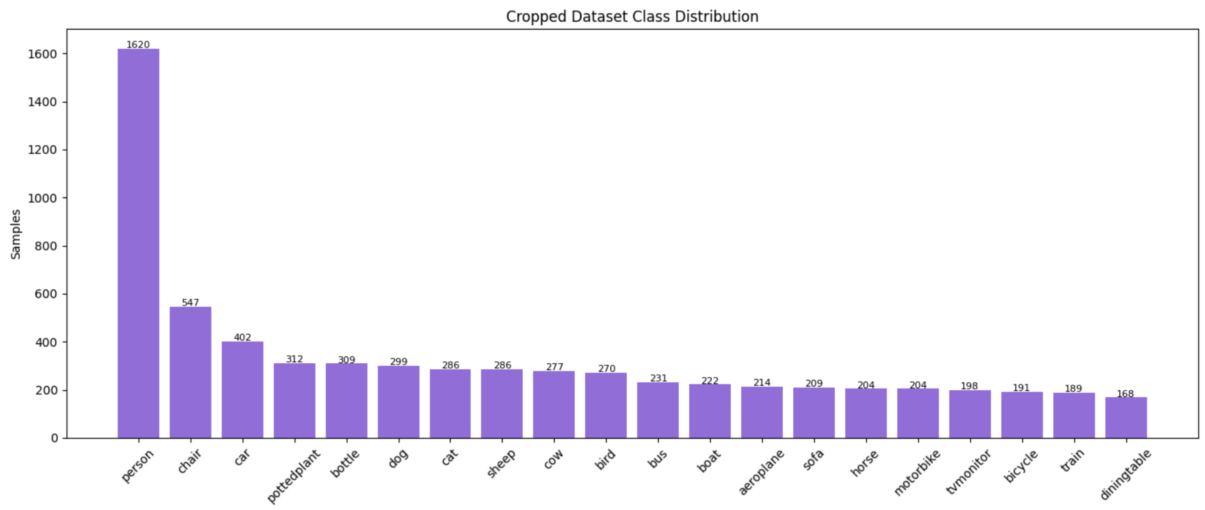
True \ Predicted	aeroplane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	diningtable	dog	horse	motorbike	person	pottedplant	sheep	sofa	train	tvmonitor
aeroplane	20	0	6	7	0	0	3	2	1	1	0	0	0	0	1	0	0	0	2	0
bicycle	2	11	3	1	0	1	2	1	0	1	0	0	2	4	2	1	2	0	4	1
bird	2	0	6	5	0	1	4	4	0	1	0	11	1	1	10	2	2	0	4	0
boat	1	0	2	23	0	1	4	1	0	0	0	0	0	0	8	1	1	0	2	0
bottle	0	0	0	2	16	1	5	5	4	1	0	2	0	0	20	2	1	0	0	3
bus	0	0	2	3	1	27	3	0	0	0	1	0	0	0	4	1	0	0	4	0
car	3	0	2	2	0	3	40	3	0	0	0	0	2	4	14	3	0	0	2	3
cat	0	1	5	2	1	0	2	17	2	2	1	7	0	1	8	4	2	0	1	1
chair	0	1	2	3	6	4	12	6	13	3	4	7	2	2	31	4	1	0	4	4
cow	1	1	0	0	0	1	1	4	1	26	0	6	1	0	4	0	7	0	1	2
diningtable	1	2	0	1	0	2	6	2	0	1	5	3	0	1	8	1	0	0	1	0
dog	0	0	1	2	2	0	1	13	0	6	0	15	0	0	10	3	3	0	3	1
horse	2	1	0	2	1	1	1	4	0	11	0	5	0	0	7	1	3	0	2	0
motorbike	2	4	1	1	0	0	5	0	0	0	0	1	0	15	8	1	1	0	2	0
person	1	0	3	10	12	3	17	9	6	2	1	15	2	4	221	8	0	0	10	0
pottedplant	0	1	0	3	1	3	3	4	0	0	0	1	0	2	9	33	0	0	2	0
sheep	0	0	2	1	0	0	1	1	0	11	1	10	1	0	6	0	19	0	4	0
sofa	0	0	0	1	1	2	11	2	2	0	2	3	0	1	8	5	0	0	3	1
train	1	0	0	1	0	8	1	0	0	1	0	0	0	1	0	0	0	0	24	1
tvmonitor	0	0	1	2	0	2	2	1	0	3	0	0	1	0	6	2	1	0	1	17

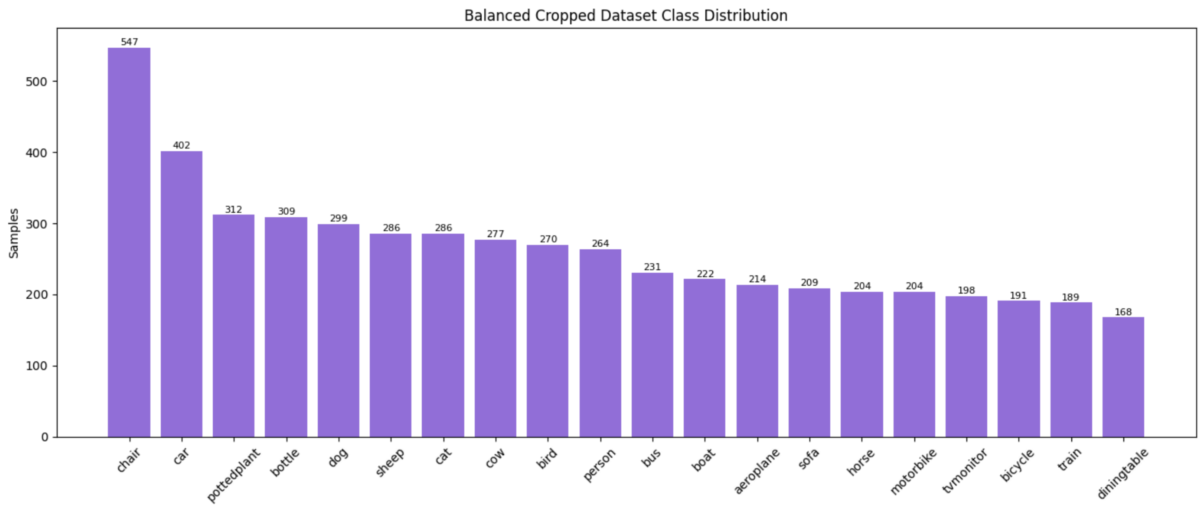
However, as you can see in the confusion matrix above, ‘person’ class had much higher frequency which resulted in us balancing out the dataset by dropping some of the ‘person’-labeled images.



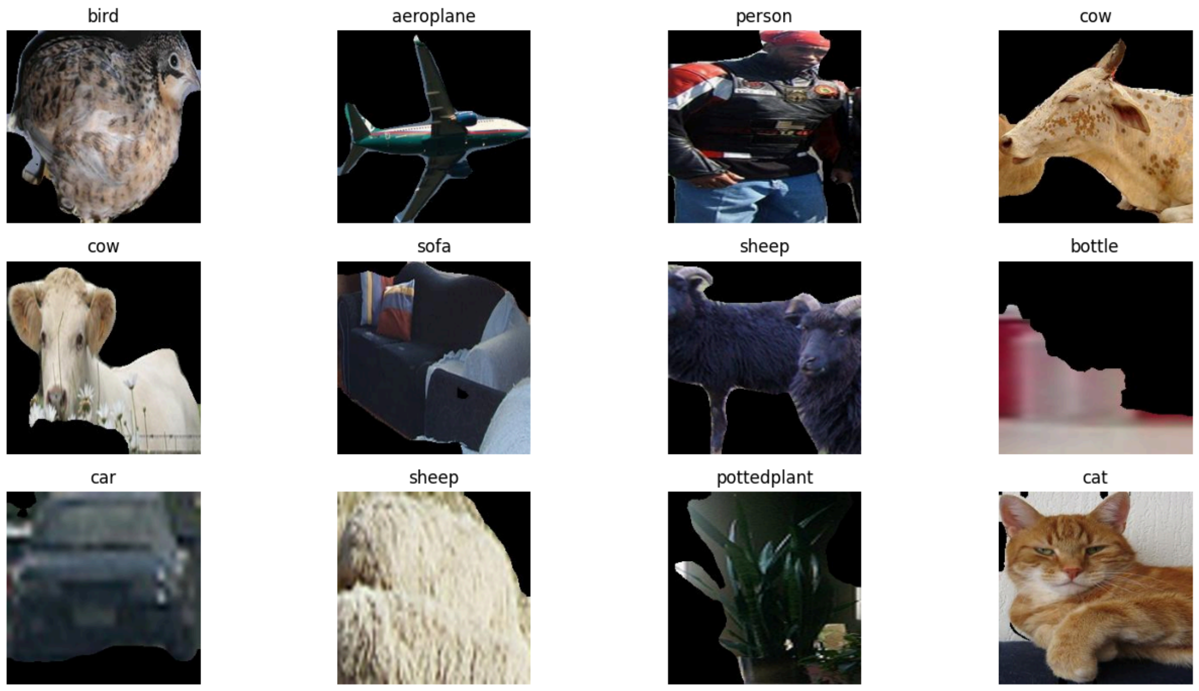
True \ Predicted	aeroplane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	diningtable	dog	horse	motorbike	pottedplant	sheep	sofa	train	tvmonitor
aeroplane	20	0	6	7	0	0	3	2	1	1	0	0	0	0	0	0	0	2	0
bicycle	2	11	3	1	0	1	2	1	0	1	0	0	2	4	1	2	0	4	1
bird	2	0	6	5	0	1	4	4	0	1	0	11	1	1	2	2	0	4	0
boat	1	0	2	23	0	1	4	1	0	0	0	0	0	0	1	1	0	2	0
bottle	0	0	0	2	16	1	5	5	4	1	0	2	0	0	2	1	0	0	3
bus	0	0	2	3	1	27	3	0	0	0	1	0	0	0	1	0	0	4	0
car	3	0	2	2	0	3	40	3	0	0	0	0	2	4	3	0	0	2	3
cat	0	1	5	2	1	0	2	17	2	2	1	7	0	1	4	2	0	1	1
chair	0	1	2	3	6	4	12	6	13	3	4	7	2	2	4	1	0	4	4
cow	1	1	0	0	0	1	1	4	1	26	0	6	1	0	0	7	0	1	2
diningtable	1	2	0	1	0	2	6	2	0	1	5	3	0	1	1	0	0	1	0
dog	0	0	1	2	2	0	1	13	0	6	0	15	0	0	3	3	0	3	1
horse	2	1	0	2	1	1	1	4	0	11	0	5	0	0	1	3	0	2	0
motorbike	2	4	1	1	0	0	5	0	0	0	0	1	0	15	1	1	0	2	0
pottedplant	0	1	0	3	1	3	3	4	0	0	0	1	0	2	33	0	0	2	0
sheep	0	0	2	1	0	0	1	1	0	11	1	10	1	0	0	19	0	4	0
sofa	0	0	0	1	1	2	11	2	2	0	2	3	0	1	5	0	0	3	1
train	1	0	0	1	0	8	1	0	0	1	0	0	0	1	0	0	0	24	1
tvmonitor	0	0	1	2	0	2	2	1	0	3	0	0	1	0	2	1	0	1	17

Original and updated class distributions are below:

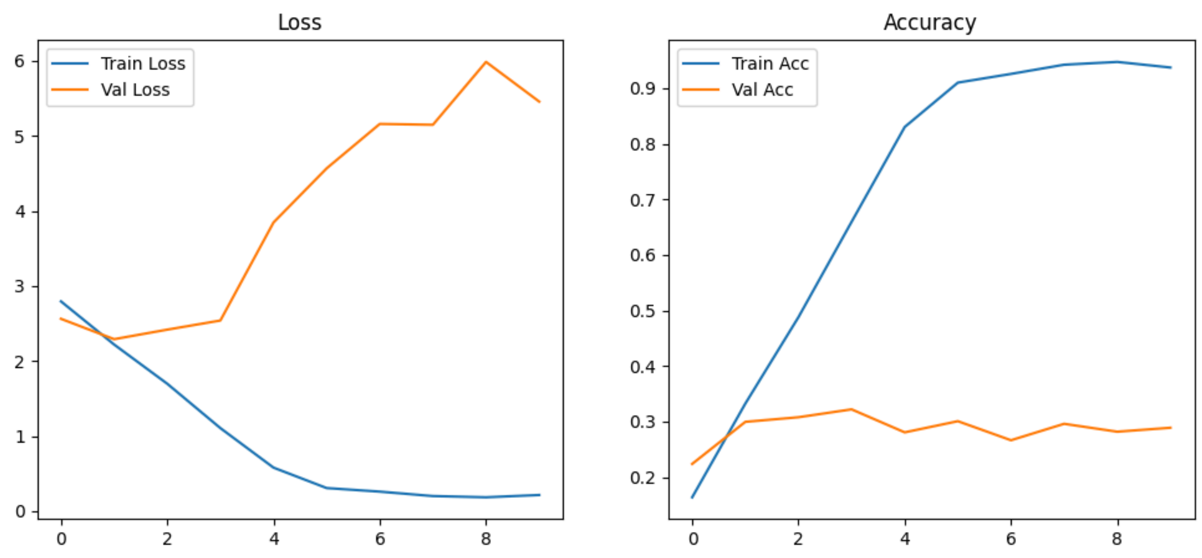


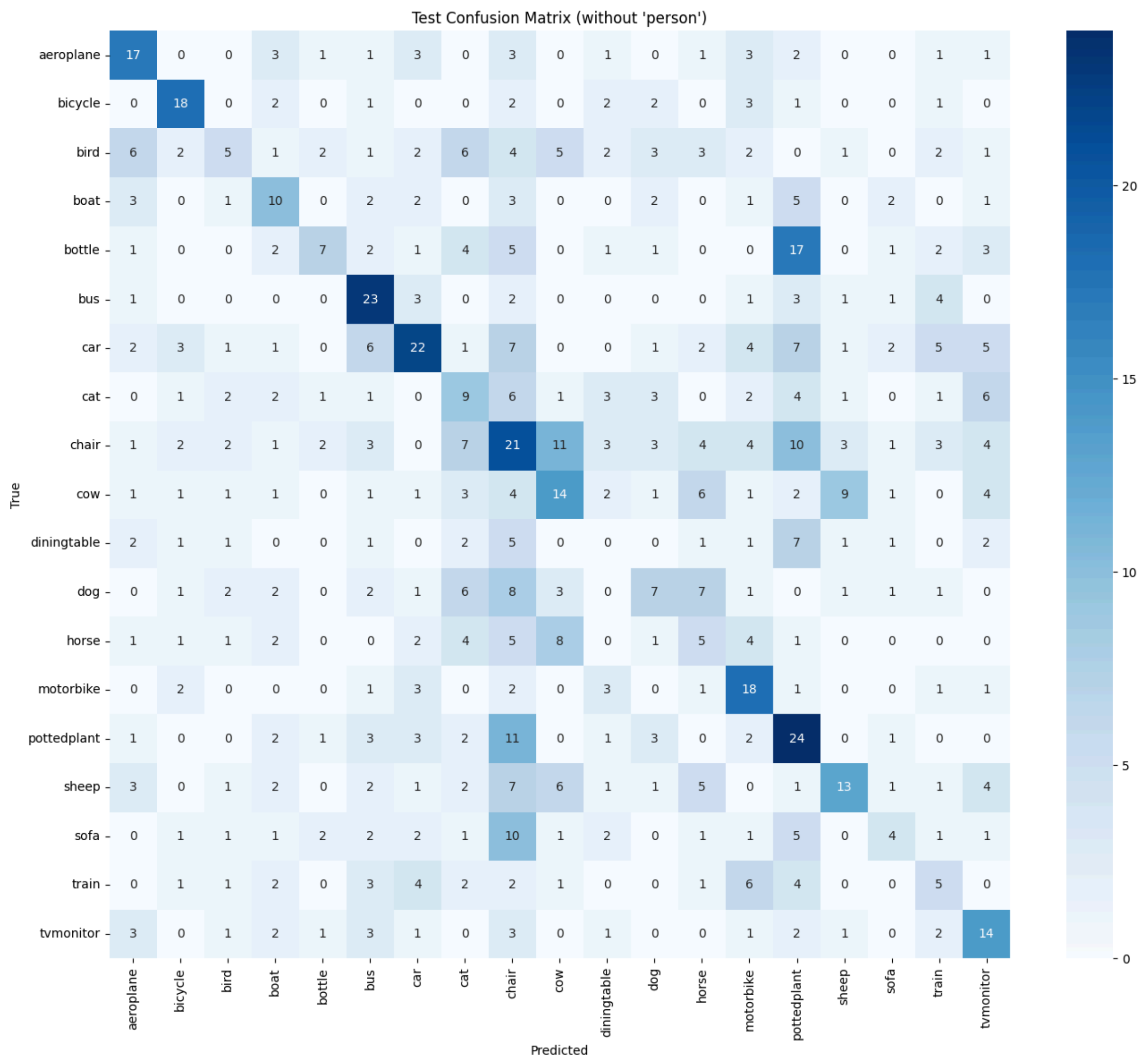


Background-foreground segmentation samples:



Training the CNN on segmented images did not affect the accuracy much:





True \ Predicted	aeroplane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	diningtable	dog	horse	motorbike	pottedplant	sheep	sofa	train	tvmonitor
aeroplane	17	0	0	3	1	1	3	0	3	0	1	0	1	3	2	0	0	1	1
bicycle	0	18	0	2	0	1	0	0	2	0	2	2	0	3	1	0	0	1	0
bird	6	2	5	1	2	1	2	6	4	5	2	3	3	2	0	1	0	2	1
boat	3	0	1	10	0	2	2	0	3	0	0	2	0	1	5	0	2	0	1
bottle	1	0	0	2	7	2	1	4	5	0	1	1	0	0	17	0	1	2	3
bus	1	0	0	0	0	23	3	0	2	0	0	0	0	1	3	1	1	4	0
car	2	3	1	1	0	6	22	1	7	0	0	1	2	4	7	1	2	5	5
cat	0	1	2	2	1	1	0	9	6	1	3	3	0	2	4	1	0	1	6
chair	1	2	2	1	2	3	0	7	21	11	3	3	4	4	10	3	1	3	4
cow	1	1	1	1	0	1	1	3	4	14	2	1	6	1	2	9	1	0	4
diningtable	2	1	1	0	0	1	0	2	5	0	0	0	1	1	7	1	1	0	2
dog	0	1	2	2	0	2	1	6	8	3	0	7	7	1	0	1	1	1	0
horse	1	1	1	2	0	0	2	4	5	8	0	1	5	4	1	0	0	0	0
motorbike	0	2	0	0	0	1	3	0	2	0	3	0	1	18	1	0	0	1	1
pottedplant	1	0	0	2	1	3	3	2	11	0	1	3	0	2	24	0	1	0	0
sheep	3	0	1	2	0	2	1	2	7	6	1	1	5	0	1	13	1	1	4
sofa	0	1	1	1	2	2	2	1	10	1	2	0	1	1	5	0	4	1	1
train	0	1	1	2	0	3	4	2	2	1	0	0	1	6	4	0	0	5	0
tvmonitor	3	0	1	2	1	3	1	0	3	0	1	0	0	1	2	1	0	2	14

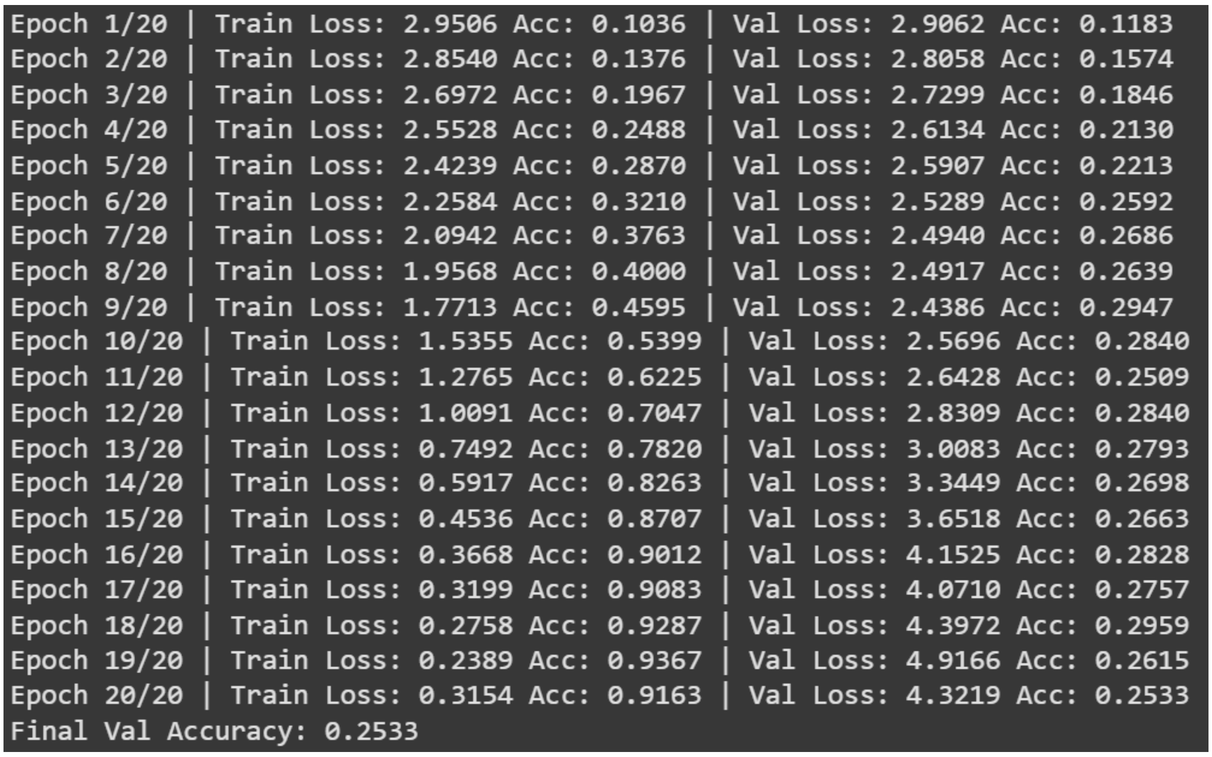
Lastly, hyperparameter search across different learning rates, optimizer and batch sizes led to the following best configuration:
{'lr': 0.0001, 'optimizer': 'adam', 'batch_size': 32}
Best Val Accuracy: 0.3432

This was the same configuration we were using by default, therefore, we did not attempt to re-run it once again.

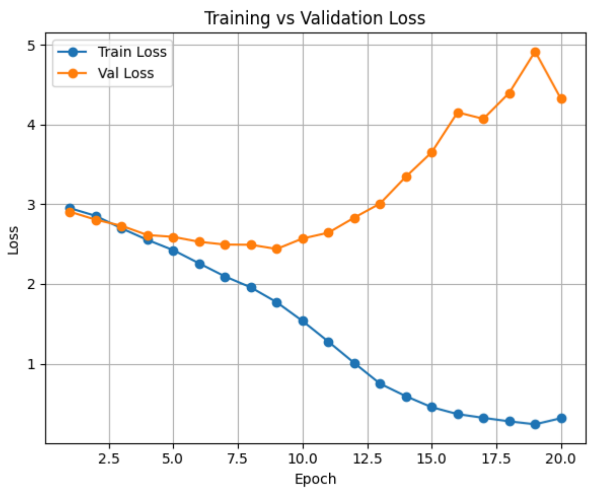
At the same time, during the hyperparameter search, we discovered this configuration results:

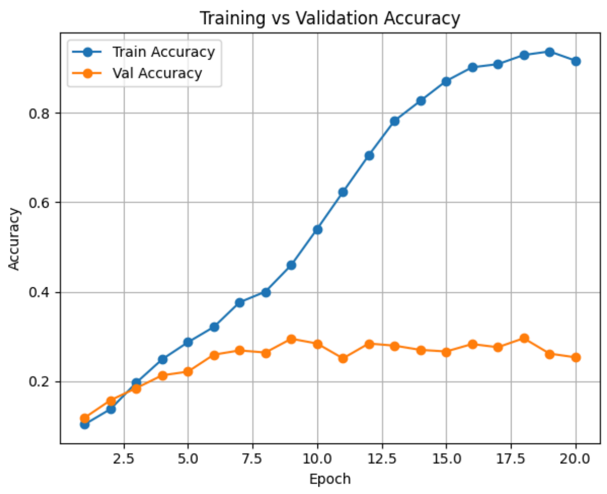
```
Trying: lr=0.001, optimizer=sgd, batch_size=32
Epoch 1/5 | Train Loss: 2.9445 Acc: 0.1044 | Val Loss: 2.9030 Acc: 0.1361
Epoch 2/5 | Train Loss: 2.8306 Acc: 0.1521 | Val Loss: 2.7969 Acc: 0.1444
Epoch 3/5 | Train Loss: 2.6687 Acc: 0.2089 | Val Loss: 2.6585 Acc: 0.1917
Epoch 4/5 | Train Loss: 2.4845 Acc: 0.2506 | Val Loss: 2.5642 Acc: 0.2225
Epoch 5/5 | Train Loss: 2.3409 Acc: 0.3006 | Val Loss: 2.4785 Acc: 0.2722
```

Looking at the results, we hypothesized that this configuration might yield better accuracy if trained on a larger number of epochs.
However, re-running this on 20 epochs led to overfitting and lower validation accuracies, debunking our hypothesis:



```
Epoch 1/20 | Train Loss: 2.9506 Acc: 0.1036 | Val Loss: 2.9062 Acc: 0.1183
Epoch 2/20 | Train Loss: 2.8540 Acc: 0.1376 | Val Loss: 2.8058 Acc: 0.1574
Epoch 3/20 | Train Loss: 2.6972 Acc: 0.1967 | Val Loss: 2.7299 Acc: 0.1846
Epoch 4/20 | Train Loss: 2.5528 Acc: 0.2488 | Val Loss: 2.6134 Acc: 0.2130
Epoch 5/20 | Train Loss: 2.4239 Acc: 0.2870 | Val Loss: 2.5907 Acc: 0.2213
Epoch 6/20 | Train Loss: 2.2584 Acc: 0.3210 | Val Loss: 2.5289 Acc: 0.2592
Epoch 7/20 | Train Loss: 2.0942 Acc: 0.3763 | Val Loss: 2.4940 Acc: 0.2686
Epoch 8/20 | Train Loss: 1.9568 Acc: 0.4000 | Val Loss: 2.4917 Acc: 0.2639
Epoch 9/20 | Train Loss: 1.7713 Acc: 0.4595 | Val Loss: 2.4386 Acc: 0.2947
Epoch 10/20 | Train Loss: 1.5355 Acc: 0.5399 | Val Loss: 2.5696 Acc: 0.2840
Epoch 11/20 | Train Loss: 1.2765 Acc: 0.6225 | Val Loss: 2.6428 Acc: 0.2509
Epoch 12/20 | Train Loss: 1.0091 Acc: 0.7047 | Val Loss: 2.8309 Acc: 0.2840
Epoch 13/20 | Train Loss: 0.7492 Acc: 0.7820 | Val Loss: 3.0083 Acc: 0.2793
Epoch 14/20 | Train Loss: 0.5917 Acc: 0.8263 | Val Loss: 3.3449 Acc: 0.2698
Epoch 15/20 | Train Loss: 0.4536 Acc: 0.8707 | Val Loss: 3.6518 Acc: 0.2663
Epoch 16/20 | Train Loss: 0.3668 Acc: 0.9012 | Val Loss: 4.1525 Acc: 0.2828
Epoch 17/20 | Train Loss: 0.3199 Acc: 0.9083 | Val Loss: 4.0710 Acc: 0.2757
Epoch 18/20 | Train Loss: 0.2758 Acc: 0.9287 | Val Loss: 4.3972 Acc: 0.2959
Epoch 19/20 | Train Loss: 0.2389 Acc: 0.9367 | Val Loss: 4.9166 Acc: 0.2615
Epoch 20/20 | Train Loss: 0.3154 Acc: 0.9163 | Val Loss: 4.3219 Acc: 0.2533
Final Val Accuracy: 0.2533
```



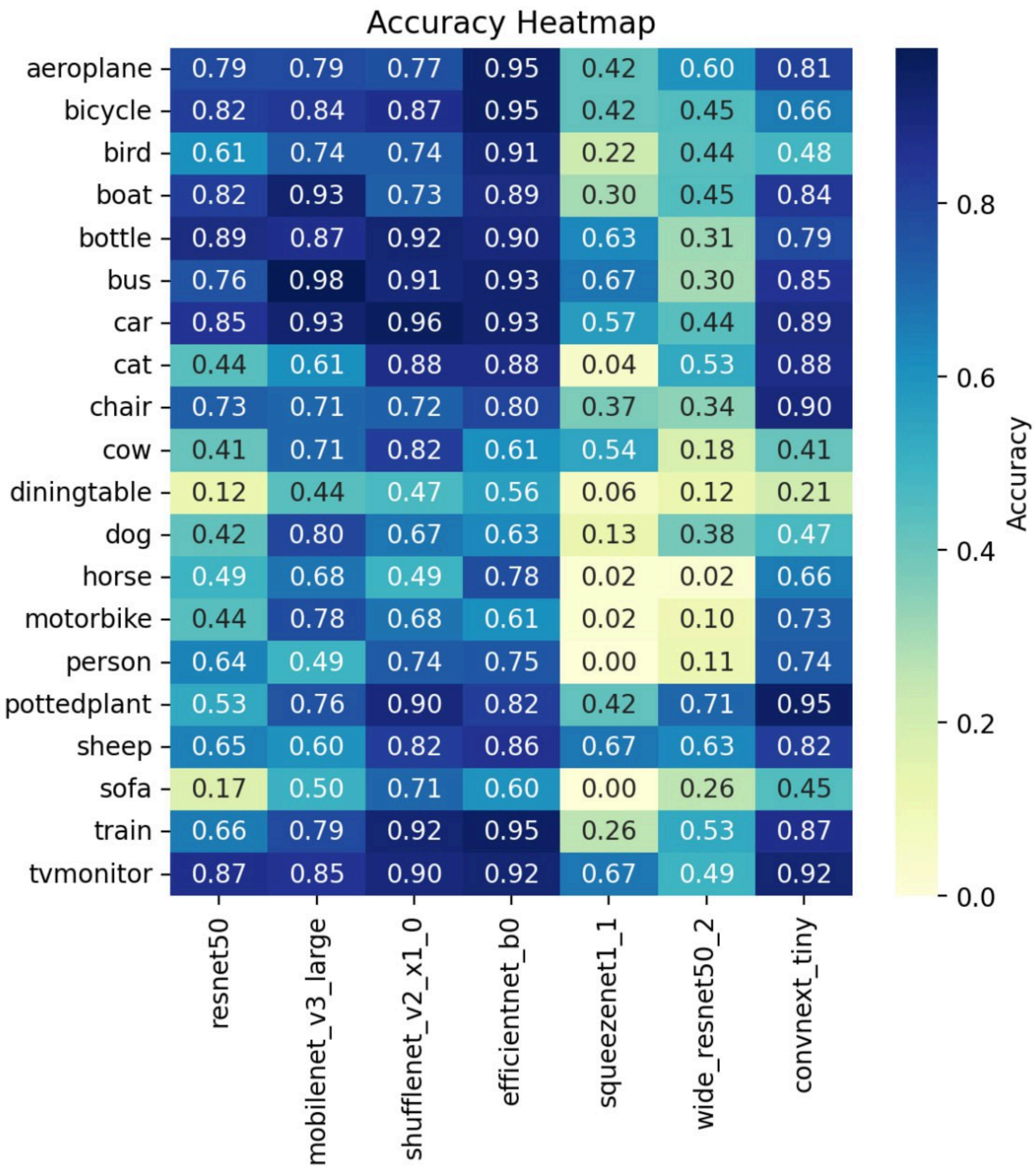


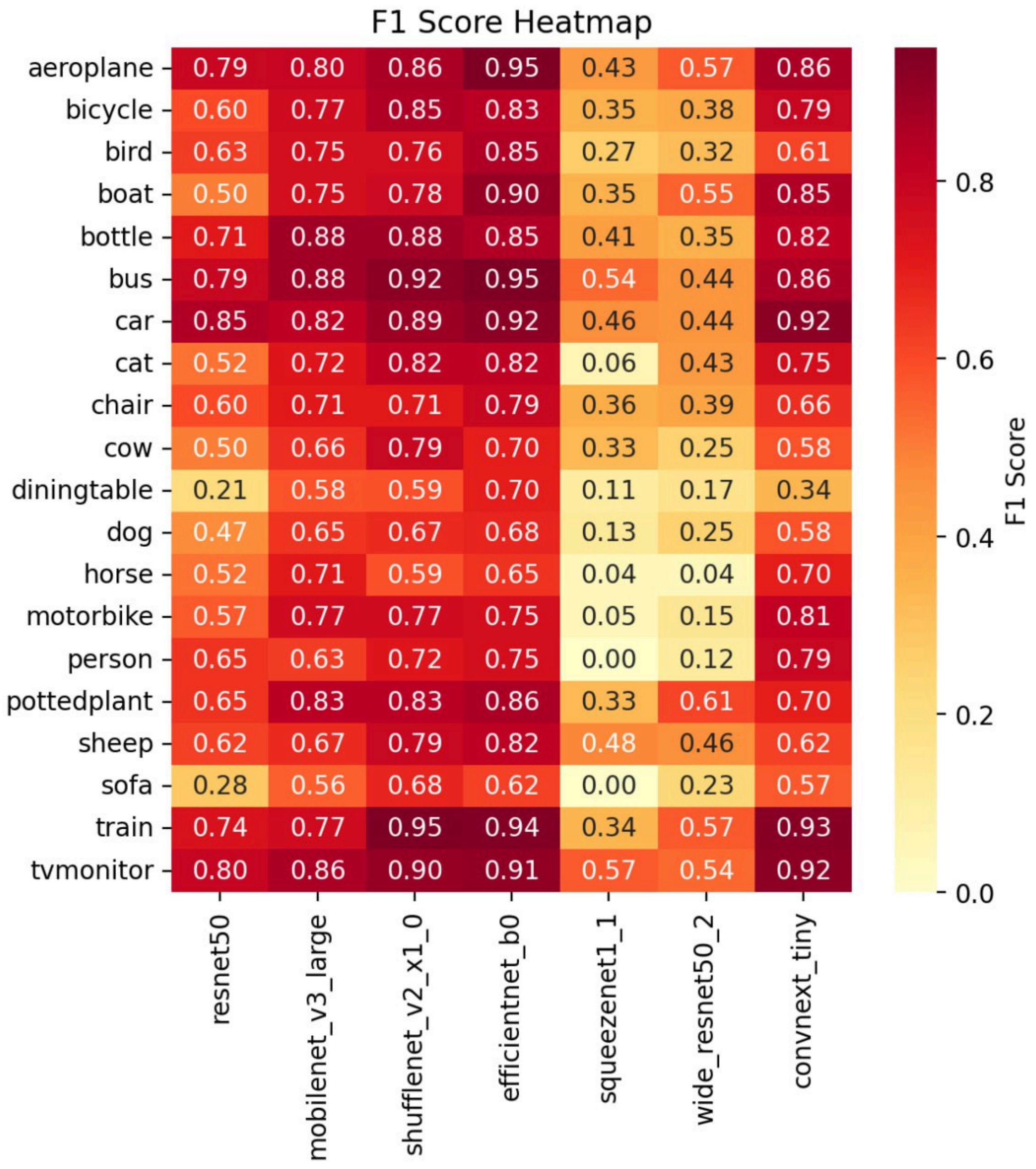
Other deep learning models parameters:

Fine-tuning Type: "Full" = all layers trainable, "Head only" = only classifier head trainable (rest frozen)
Optimizer: `Adam` (no momentum) and `SGD` (momentum=0.9)
Weight Decay: `1e-4` (for regularization, in both Adam and SGD) (avoid overfitting)
Learning Rate: `1e-3` for Adam (classic starting point), `1e-2` for SGD (slightly higher, as is standard)
Train/Valid/Test: 64/16/14 -> stratified and balanced dataset
Batch_size: 32
Epochs: 10
Scheduler: None

Characteristics of models, their params, and small comments

Model Name	**Typical Input Size**	**# Params (approx)**	**Comments**
SimpleCNN	224x224	<1M	Your custom small CNN
ResNet50	224x224	25M	Deep, standard for transfer learning
Wide ResNet50_2	224x224	68M	Same as ResNet50 but wider; better accuracy
MobileNetV3 Large	224x224	5.5M	Very efficient, fast, great for mobile/embedded
EfficientNet B0	224x224	5.3M	State-of-the-art efficiency/accuracy tradeoff
ShuffleNet v2 x1.0	224x224	2.3M	Extremely lightweight, designed for speed
ConvNeXt Tiny	224x224	28M	Modern "ResNet-meets-ViT", top accuracy, efficient blocks
SqueezeNet 1.1	224x224	1.25M	One of the smallest deep models; uses "fire modules"
ViT (vit_base_patch16_224)	224x224	86M	Vision Transformer, pure transformer, high data requirement
Swin (swin_tiny_patch4...)	224x224	28M	Hierarchical transformer; windowed attention
GoogLeNet (Inception v1)	224x224	6.6M	Older, popular for fast training, multiple paths
Inception v3	299x299*	24M	For 224x224 inputs, works but not optimal; original is 299x299


Accuracy Heatmap
aeroplane 0.79 0.79 0.77 0.95 0.42 0.60 0.81
bicycle 0.82 0.84 0.87 0.95 0.42 0.45 0.66
bird 0.61 0.74 0.74 0.91 0.22 0.44 0.48
boat 0.82 0.93 0.73 0.89 0.30 0.45 0.84
bottle 0.89 0.87 0.92 0.90 0.63 0.31 0.79
bus 0.76 0.98 0.91 0.93 0.67 0.30 0.85
car 0.85 0.93 0.96 0.93 0.57 0.44 0.89
cat 0.44 0.61 0.88 0.88 0.04 0.53 0.88
chair 0.73 0.71 0.72 0.80 0.37 0.34 0.90
cow 0.41 0.71 0.82 0.61 0.54 0.18 0.41
diningtable 0.12 0.44 0.47 0.56 0.06 0.12 0.21
dog 0.42 0.80 0.67 0.63 0.13 0.38 0.47
horse 0.49 0.68 0.49 0.78 0.02 0.02 0.66
motorbike 0.44 0.78 0.68 0.61 0.02 0.10 0.73
person 0.64 0.49 0.74 0.75 0.00 0.11 0.74
pottedplant 0.53 0.76 0.90 0.82 0.42 0.71 0.95
sheep 0.65 0.60 0.82 0.86 0.67 0.63 0.82
sofa 0.17 0.50 0.71 0.60 0.00 0.26 0.45
train 0.66 0.79 0.92 0.95 0.26 0.53 0.87
tvmonitor 0.87 0.85 0.90 0.92 0.67 0.49 0.92
resnet50
mobilenet_v3_large
shufflenet_v2_x1_0
efficientnet_b0
squeezenet1_1
wide_resnet50_2
convnext_tiny
Accuracy
0.0
0.2
0.4
0.6
0.8



VII. Conclusion

Highest performance metrics across all models:

Model: mobilenet_v3_large_head_only
Test accuracy: 0.8231

Model: efficientnet_b0_head_only
Test accuracy: 0.8098

Model: shufflenet_v2_x1_0_head_only
Test accuracy: 0.7039

Model: mobilenet_v3_large_adam
Test accuracy: 0.8278